

REMARKS

Claim 1 has been rewritten as claim 40 and now incorporates the subject-matter of claim 3. Claims 15, 26 and 30 have been rewritten in independent form as claims 51, 61 and 65, respectively. In addition, new claims 71 and 72 have been added so as to cover the embodiment described as "Example 4" on pages 16 and 17 of the application.

All claims stand rejected under 35 USC 103 over Beelitz and Matyas.

Matyas describes a method of software protection intended to discourage the copying of software by allowing an encrypted program to be executed only on a designated computer. Each computer program sold by a software vendor is encrypted using a file encryption key. A user, upon purchasing the computer program, is required to telephone the vendor in order to obtain a password. The user must then provide information which includes, among other things, a computer number that is unique to the computer on which the computer program is to be installed (page 5, lines 14-20 of Matyas). This information is then used to encrypt the encryption key that was used to encrypt the computer program. The encrypted encryption key is then provided by the vendor to the user as a password (page 5, lines 36-43 of Matyas). When the computer program is executed by the user for the first time, the computer program prompts the user for the password. The password, once input, is then written to the header part of the computer program so that the user is not prompted in future. During subsequent execution of the computer program, the unique computer number of the computer is used to decrypt the password in order to obtain the encryption key. The encrypted computer program is then decrypted using the encryption key. Since the password can only be decrypted using a unique computer number, the computer program can only be decrypted and executed on the computer registered with the vendor (page 6, lines 31-40 of Matyas).

Beelitz describes a method of preventing unauthorized access to a computer program. This is achieved by means of a routine which is executed whenever the computer program is executed. The routine checks a predetermined memory location to determine whether the processor executing the computer program is operating in a suspended mode. If the processor is operating in a suspended mode, the routine

stops the processor from executing the computer program. The memory location checked by the routine is the interrupt register and/or flag register of the processor. If the breakpoint flag of the interrupt register points to a routine (such as that used by a debug program), or if the trap flag of the flag register is set (indicating single-step mode), then the routine terminates execution of the computer program (page 2, paragraphs 2 and 3, and page 4, paragraph 2 of Beelitz). Beelitz therefore describes a method of preventing unauthorized access of a computer program by a debug program by providing a separate routine that monitors registers of the processor executing the program.

Applicant submits that each of the independent claims is patentable over Beelitz and Matyas, whether taken singly or in combination. Applicant will consider the current independent claims in order.

#### Claim 40

With the method of Beelitz, a copy of the computer program is held in non-protected memory during execution. Accordingly, it is possible to dump the contents of the memory to disk in order to obtain a copy of the computer program without the inclusion of the anti-debug routine.

With the method of Matyas, each computer is provided with a cryptographic facility that includes, among other things, protected RAM into which the decrypted program code is written. This protected RAM then provides access to the decrypted program code for execution purposes only, i.e. access for non-execution purposes is prevented. Whilst the provision of protected RAM hinders the dumping of the program code, the method described by Matyas suffers from the very serious drawback that each computer must be provided with a cryptographic facility having protected memory. The method is not therefore suited to off-the-shelf computers. Additionally, should a method of accessing the protected memory for non-execution purposes be found, a full copy of the program code could then be dumped to disk.

Claim 40 covers the embodiment described as "Example 2" on pages 9 and 10 of the specification. The claim is intended to cover an executable file (digital data arrangement) that includes program code and security code. The program code is protected to prevent copying,

hence it is referred to in the description and claims as protected code. The program code is initially incomplete, such that if someone were to dump the program code from memory to disk, they would not obtain a complete copy of the program code. For each missing part of code, the program code includes a call instruction to the security code. When called, the security code replaces the call instruction with a corresponding missing part of executable code. Only once each and every call instruction has been executed is a complete copy of the program code obtained. Accordingly, unlike the prior art, a complete copy of the program code is obtained only during execution of the program code.

Although claim 40 refers to one or more call instructions, in practice the program code will include hundreds of call instructions, many of which will be associated with event-driven routines, i.e. routines that are executed only in response to some external trigger such as a user input. For example, a word processing program will have event-driven routines for creating a new document, printing a document, reporting errors etc. Indeed, every option that is available to a user and every possible outcome will have one or more event-driven routines associated therewith. Accordingly, it will not be possible for a hacker to obtain a complete copy of the program code unless each event-driven routine has been executed, i.e. until all possible paths of the program code have been exercised. For a computer program having many hundreds of event-driven routines, it is prohibitively expensive in terms of time and resources to exercise all control paths.

Neither Matyas nor Beelitz describes an arrangement in which protected program code includes call instructions to security code which, when executed, causes the call instruction to be replaced by executable code. It is therefore submitted that the invention defined in claim 40 is not disclosed or suggested by Matyas and Beelitz, whether taken singly or in combination.

#### Claim 51

With the method of Beelitz, the protected program is preferably stored on a hard disk as a compressed and encrypted executable such that a user cannot simply access and decompile the program (page 3, final paragraph of Beelitz). The executable comprises a header and a

copy of the protected program in compressed and encrypted form. The header includes instructions for decompressing and decrypting the protected program and for loading the decrypted program into memory. A problem with the method of Beelitz is that the header is vulnerable to attack by a hacker. In particular, a hacker may change the header instructions such that a copy of the program, when decrypted, is written to disk rather than to memory.

With the method of Matyas, the protected program is provided in encrypted form and includes a header. Unlike Beelitz, however, the header does not include decryption instructions. Instead, the decryption instructions are held within ROM of a cryptographic facility. Whilst this improves security and prevents surreptitious code from being inserted into the decryption instructions, Matyas suffers from the drawback that every computer must be provided with a cryptographic facility having decryption instructions provided in ROM. Consequently, the method of Matyas is not suitable for protecting programs to be executed on off-the-shelf computers.

Claim 51 is directed to a data arrangement in which the decryption instructions are initially provided in a non-executable form. The instructions are then converted, during subsequent use, into an executable form by means of an algorithm that employs a conversion key. Importantly, the conversion key is derived from a block of data within the data arrangement. Accordingly, the decryption instructions are converted into executable form and the protected data is decrypted only in the event that no corruption to the data arrangement is detected. Should any surreptitious code be inserted into the arrangement (e.g. debug code or virus code), the decryption instructions would not be converted into an executable form and the protected data would not be decrypted.

Neither Matyas nor Beelitz describe an arrangement in which decryption instructions for decrypting protected data are provided initially in a non-executable form. Moreover, there is no suggestion of converting the decryption instructions into an executable form using a conversion key derived from a block of data of the arrangement. It is therefore submitted that the invention defined in claim 51 is not disclosed or suggested by Matyas and Beelitz, whether taken singly or in combination.

## Claim 61

Beelitz describes a method in which an anti-debug routine, separate to the computer program under protection, monitors register locations of a processor. Accordingly, should any debug code be inserted into the computer program, the anti-debug routine would detect execution of the debug code and terminate the computer program. However, the anti-debug routine is not capable of monitoring whether other forms of surreptitious code have been embedded in the computer program. For example, virus code inserted into the computer program would not be detected by the anti-debug routine. Additionally, the computer program might include security code for checking software license details. Code could be injected into the computer program that circumvents the security code, which would not be detected by the anti-debug routine.

Matyas, on the other hand, decrypts the computer program into protected RAM that provides access to the decrypted program code for execution purposes only. Whilst the provision of protected RAM prevents surreptitious code from being injected into the computer program, Matyas suffers from the previously-identified problem that each computer must be provided with a cryptographic facility having protected memory. The method described by Matyas is not therefore suitable for the protection of programs to be executed on off-the-shelf computers.

Claim 61 is directed to an arrangement comprising executable code that, when executed, creates protected code. The protected code comprises a plurality of executable steps which are written in a different order each time the protected code is created. Consequently, any surreptitious code injected into the protected code will be overwritten next time the executable code is executed. Moreover, if one of the steps of the protected code relates to a security check, any attempt to circumvent this step (e.g. by including jump instructions) will prove ineffective since the location of the security check will vary with each execution.

Neither Matyas nor Beelitz describes an arrangement in which executable code creates protected code having a plurality of executable steps that are written in a different order each time the executable code is executed. Instead, both Matyas and Beelitz describe conventional methods of compression and decryption in which

the protected code, upon decryption, are written to memory in the same order each time. It is therefore submitted that the invention defined in claim 61 is not disclosed or suggested by Matyas and Beelitz, whether taken singly or in combination.

#### Claim 65

As identified above in connection with claim 40, Beelitz describes a method in which the computer program is held in non-protected memory during execution. Consequently, it is possible to dump the contents of the memory to disk in order to obtain a copy of the computer program.

With the method of Matyas, each computer is provided with a cryptographic facility that includes protected RAM into which the decrypted program code is written. Whilst the provision of protected RAM hinders the dumping of the program code, the method described by Matyas is not suitable for protecting programs to be executed on off-the-shelf computers which are not provided with protected memory. Additionally, should a method of accessing the protected memory be found, a full copy of the program code could then be dumped to disk.

Claim 65 is directed to executable code which, when executed, provides only one part of protected code at any one time. In particular, a first part of the protected code is created and executed, and then a second part of the protected code is created and executed. Importantly, the first part of the protected code is corrupted upon creation of the second part of the protected code (e.g. the second part overwrites the first part). Consequently, at no time is a full copy of the protected code made available. It is not therefore possible to perform a memory dump in order to obtain a complete copy of the protected code.

Neither Matyas nor Beelitz describe executable code that creates and executes only one part of protected code at any one time, whilst also ensuring that each part of the protected code is corrupted upon creation and execution of a new part of the protected code. It is therefore submitted that the invention defined in claim 65 is not disclosed or suggested by Matyas and Beelitz, whether taken singly or in combination.

## Claim 36

This claim corresponds substantially to claim 51. It is therefore submitted that, for the reasons provided above in connection with claim 51, the invention defined in claim 36 is not disclosed or suggested by Matyas and Beelitz, whether taken singly or in combination.

## Claim 71

As noted above in connection with claim 61, the anti-debug routine of Beelitz is capable only of detecting whether debug code has been injected into the computer program. The anti-debug routine is not, however, capable of monitoring whether other forms of surreptitious code have been embedded in the computer program, e.g. virus code or code intended to circumvent security checks.

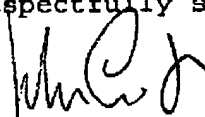
Matyas describes a method in which a protected computer program is decrypted into protected memory. Again, whilst the provision of protected RAM prevents surreptitious code from being injected into the computer program, the method described by Matyas is not therefore suitable for the protection of programs to be executed on off-the-shelf computers. Moreover, if the method described by Matyas were implemented using non-protected memory or if a method were found to access the protected memory for non-executable purposes, surreptitious code could easily be injected into the protected program. Matyas fails to describe any method of protecting a computer program against corruption.

Claim 71 is directed to an arrangement in which the protected program code comprises a call instruction to security code. The security code, when executed, checks the integrity of the protected code and determines whether the protected code is corrupt. Corruption is detected in the event that the protected code has changed, e.g. if virus code, debug code, or code intended to circumvent security checks has been injected into the protected code. If the protected code is not corrupt, relocation code is then executed. The relocation code moves the location of the security code and then modifies the call instruction such that the call instruction now points to the new location of the security code. Next time the call instruction is executed, the security code at the new location is executed.

With the arrangement of claim 71, the call instruction to the security code is constantly changing during execution of the protected code. This in turn makes it much more difficult for a hacker or virus to recognize and disable the call instruction so as to circumvent the security code.

As already noted, Matyas fails to teach or suggest a method of protecting against corruption. Additionally, Beelitz describes an anti-debug routine which monitors registers of a processor, and is therefore able to determine whether debug code has been. Beelitz fails to teach or suggest protected program code that includes instructions to a security code, which determines whether the. Moreover, Beelitz fails to teach or suggest an arrangement in which the location of the security code is constantly changing during execution of the protected code. It is therefore submitted that the invention defined in claim 71 is not disclosed or suggested by Matyas and Beelitz, whether taken singly or in combination.

Respectfully submitted,



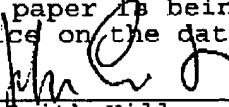
John Smith-Hill  
Reg. No. 27,730

SMITH-HILL & BEDELL, P.C.  
16100 N.W. Cornell Road, Suite 220  
Beaverton, Oregon 97006

Tel. (503) 574-3100  
Fax (503) 574-3197  
Docket: FORT 2277

Certificate of Facsimile Transmission

I hereby certify that this paper is being facsimile transmitted to the Patent and Trademark Office on the date shown below.



John Smith-Hill

12/28/05

Date